**Mountain Paths (Part 1)**

**Points**

|  |  |
| --- | --- |
| **Points** |  |
| 5 | Design |
| 80 | Runs correctly |
| 15 | Code Review |
| **100** | **TOTAL** |

**Objectives**

* 2d arrays
  + Store
  + Use
* Nested Loops
* Parallel data structures (i.e. parallel arrays … called multiple arrays in the zyBook)
* Transform data
* Read from files
* Write to files
* structs

**Submission**

**Source Code**: Submit the source code to Mimir

* + main.cpp
  + functions.h
    - Note: If you work in an IDE that names header files with .hpp instead of .h, you will need to rename functions.hpp to functions.h before submission.
  + functions.cpp

**Background**

There are many contexts in which you may want to know the most efficient way to travel over land. When traveling through mountains (let's say you're walking), perhaps you want to take the route that requires the least total change in elevation with each step you take — call it the path of least resistance. Given some topographic data, it should be possible to calculate a "greedy lowest-elevation-change walk" from one side of a map to the other.

In part I of this homework, you will implement code to read the topographical data and manipulate it so that it can be displayed. In part II (next homework), you will implement the code to identify and display certain paths.

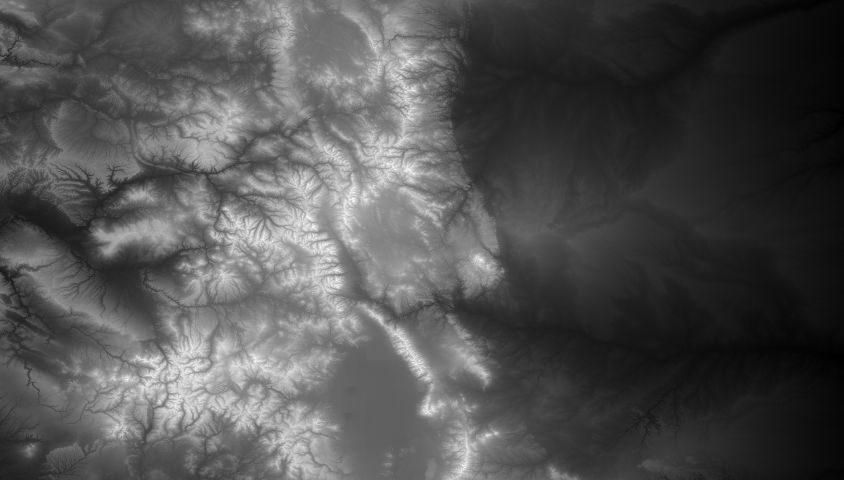
**Specifications**

**First, we Design**

As you become experienced you may not always choose to do flowcharts. Regardless, you should sketch out algorithms with pseudocode or a flowchart.

In this homework, you will read a set of topographic (land elevation) data into a 2D array and manipulate the data so that it can be visualized. In the next homework, you will extend your code so that it computes some paths through the topographic terrain (mountains) as well as visualize those paths.

**Program**

The minimum requirements for this homework are that you write code to read elevation data and produce a file representing an image that, when visualized, displays each position on the map with a color that represents the position’s elevation. The visualization will produce a picture like the one below:

***Code Requirements***

* Start with this code: [mtnpathstart.zip](https://drive.google.com/file/d/1LSKYItsnvT9-ybuLXmtYEYOSU1nRmcBB/view)
  + Do not modify the function signatures provided.
* Do not #include<vector> or #include<algorithm>

***Program Flow***

1. Read the data into a 2D array
2. Find min and max elevation to correspond to darkest and brightest color, respectively
3. Compute the shade of gray for each cell in the map
4. Produce the output file in the specified format (PPM)
5. Use an online free tool to convert your PPM file into a JPG file to visually check your results

**Step 1 - Read the data into a 2D array**

Your first goal is to read the values into a 2D array (i.e., a matrix, elevations in the start code) of *ints* from a data file.

This data file contains, in plain text, data representing the average elevations of patches of land (roughly 700x700 meters) in the US. The user of your program will provide as input three values from standard in:

1. The number of rows in the map; *(The height of the image to be produced.)*
2. The number of columns in the map; *(The width of the image to be produced.)*
3. The name of the file containing the map data.

You should validate the inputs. If there is an error, output the error as formatted below and exit the program immediately.

* Error: Problem reading in rows and columns.
* Error: Unable to open file <filename>.

After reading these data items from the user, you should implement a function to output the elevation data to check that you read the values correctly. We provide several sample data files containing elevation data for most of the state of Colorado (mountains!). Other input files can be obtained from the National Oceanic and Atmospheric Administration (<http://maps.ngdc.noaa.gov/viewers/wcs-client/>).

A data file comes as one large, space-separated list of integers. A file representing a  480-row by 844-col grid contains 405,120 integers (480\*844). Each integer is the average elevation in meters of each cell in the grid. The data is listed in row-major order, i.e., first the 844 numbers for row 0, then the 844 numbers for row 1, etc.

You should validate that you got the correct number of values from the file. If there is an error, output the error as formatted below and exit the program immediately.

* Error: Read a non-integer value.
* Error: Problem reading the file.
* Error: End of file reached prior to getting all the required data.
* Error: Too many data points.

Hints/Warnings:

* The data is given as a continuous stream of numbers - there may be no line breaks in the file.
* You may want to make sure that you are reading the data correctly before you move to the next step of this project. Implement a function to print your map data to standard out and try out your code with a file containing a small map first (you can create an input file corresponding to 4 rows and 4 columns, for example.) Then check what happens when you read the sample map data files: (100 by 100, 480 by 480, 844 by 480).
* Note that failbit and eofbit can be on at the same time. Just because eofbit is set, does not mean that the previous read did not succeed.
* Elevations can be negative if they are below sea level.
* You can download the example map data files (shown below) packaged together (in zip format) via:
  + [map-inputs.zip](https://drive.google.com/file/d/1dL4Wsrpu4xpjZ1GVPqogXgcxxIVlv60Y/view)(note: will need to be logged into TAMU Google account for access).

|  |  |
| --- | --- |
| **Download Link for Map Data Files** | **Resulting Image** |
| 100X100 | https://lh3.googleusercontent.com/UrhLygL63fF-aEct80R-oCjOb3LhGtc89YTHlEamzZYcwDIPEFsC4hJhc1Dy4E6yia2Q8kBcYT1dNEC91Lg5xSrx3Kgn9oBXjbQk0D4RQUlE31PwCV4FPp01D_GVTIymHMq35iIb |
| 480X480 | https://lh3.googleusercontent.com/5VmmSaTaAjioB-6GQWi9LdVc78rZsPyAAp44IVUv2XC_gHoVrKq9NiVhxyjyxlEb6Dm222WwYQWlKABwdV32fJBnp8kqSV6Dz7FXWulmwQXosN3gCCHZUOLb03uM_okWahn6TKOd |
| 87X189 | https://lh4.googleusercontent.com/EKGwcny53EFz2LbL37COda2f56NL51sqKnm6nZEE3oeuPyQXEulgBXZ78LJGpZmSeGywdWpRp-BPIC-_7Yr6LQrFjVsP5yvm9VzAbs8r-pgxHPiNY-OsrEdPS2pbgLNUvhHeOs4r |
| 844X480 | https://lh6.googleusercontent.com/8LS2ET7I1tO8sRwqGtfdVUYaJ4yUjKyn-sBy1tNz0r2xc1YbT7Ll2G9BqU3wp19uGoEgx0Z8q2Umehjoj03tzFw3k_5EdMnn2WFnaV0gCcOOX0k47x_S6ZAvKwHPlz3UX8ftH_6Z |

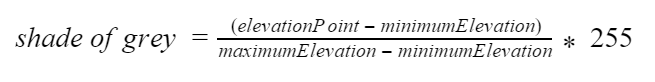
Step 2 - Find the min and max values

In order to draw the map with a grayscale that indicates the elevation, you will need to determine which scale to use, i.e., which will be shown as a dark area (with low elevation) and which ones will be shown as bright areas (high elevation).  This means you need to find the min and max values in the map first, so that you know which values to associate with the brightest and darkest colors. You can compute the min and max elevation values by writing the **findMaxMin** function that scans the entire map data and keeps track of the smallest and largest values found so far.

Test this functionality to make sure you’re getting the correct values for min and max before you proceed to implement the rest of the program. You may want to check with a friend or colleagues in the Piazza forum to see if other students are getting the same min and max values for the provided input files.

Step 3 - Compute the color for each part of the map and store

For each value in the array of elevation data, you will calculate the shade of grey that corresponds to that elevation point using the **scaleValue** function that uses the following equation:



Use the min and max elevations calculated from Step 2. Also, **round** the value you get to the nearest whole number since RGB values are integers.

Implement the **loadGreyscale** function to go through all of the elevations and put the greyscale value into the parallel image array. Each element of the Pixel struct (red, green, and blue) should be assigned the same value since using the same value for R, G and B will result in grey.

The structure of the array should mirror the array with the elevation data. This structure is required in the next part of the homework. If you use an alternative structure, then you will have to redo that part in the next homework. Search the zyBook for more information about parallel arrays searching for “multiple arrays”. Unfortunately, the zyBook uses this more ambiguous term.

Step 4 - Produce the output file in the PPM format

PPM (portable pixel map) format is a specification for representing images using the RGB color model. PPM is not used widely because it is very inefficient (for example, it does not apply any data compression to reduce the space required to represent an image.) But PPM is very simple, and there are programs available for Windows, Mac, and Linux that can be used to view ppm images. Even more conveniently, you can use an online tool with your browser to convert a PPM file into a widely used format such as JPG. We will be using Plain PPM, which has the information in text format (readable as decimal numbers instead of binary.)

You will implement the **outputImage** function to write the PPM data to the output file.

In Step 3, you computed the RGB values for the shades of gray. All you will need to do to get a PPM image for these RGB values is to write a preamble before writing the RGB numbers into a file. We will follow the convention that if the input file is named *input.dat*, then your program will generate a file named *input.dat.ppm*

A PPM file has the following format:

* First line: string “P3”
* Second line: width (number of columns) and height (number of rows)
* Third line: max color value (for us, 255)
* Rest of the file: list of RGB values for the image, expressed as a raster of rows, from top to bottom. Each row contains the RGB values (i.e., three values) for each column.

Example:  
Note: We have added colors showing that every three numbers represent a single pixel.

|  |
| --- |
| P3 4 4 255 0 0 0 255 0 0 0 0 0 0 255 0   255 255 255 255 0 255 0 0 0 0 255 0   255 255 0 0 0 255 125 0 255 255 0 125 0 0 255 255 255 0 125 125 125 239 239 239 |

This same file could also look like (all pixels on a single line):

|  |
| --- |
| P3 4 4 255 0 0 0 255 0 0 0 0 0 0 255 0 255 255 255 255 0 255 0 0 0 0 255 0 255 255 0 0 0 255 125 0 255 255 0 125 0 0 255 255 255 0 125 125 125 239 239 239 |

Or even this: (But this one makes no sense to a human reading it.)

|  |
| --- |
| P3 4 4 255 0 0 0 255 0 0  0 0 0 0 255  0 255 255 255 255  0 255 0 0 0  0 255 0 255 255  0 0 0 255 125  0 255 255 0 125  0 0 255 255 255  0 125 125 125 239  239 239 |

Sample PPM File:

[blocks.ppm](http://courses.cse.tamu.edu/jmichael/f16/121/homework/5/blocks.ppm) https://lh5.googleusercontent.com/7Ptv6gg0M721Gz5Tl50V7hn1SoRtsW13rzh2etWPXJ-wF3sDtbnREREj8mi552m2I7GHBNEXtV7SUdmqIlMxJLqVPPZQeZp7LmxI1XUUuxk3bIq-j2o_CaFW5avEj8zHTS6L0niR

***Step 5 - Use an online free tool to visualize the PPM file you generated***

In this step you do not write any code: you simply use a tool to convert the .ppm file into a .jpg file. The tool is available at:

[https://www.coolutils.com/online/PPM-to-PNG#](https://www.coolutils.com/online/PPM-to-PNG)

The usage is very intuitive: 